Gestion des erreurs temporelles dans un système temps réel critique

Etienne Borde

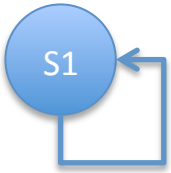
etienne.borde@telecom-paristech.fr

Introduction

Erreur temporelle: pourquoi?

- WCET mal calculé (caches/multi-cœurs/pagination…)
  - ⌘ en pratique, les industriels sont peu enclins à utiliser des WCET calculés pour le dimensionnement de leurs applications… Il préfèreront la simulation
- Blocking time: complexité vs. Pessimisme des configurations à analyser
  - ⌘ nous avons vu au TP précédent qu'il peut être difficile d'isoler une configuration qui prouve l'ordonnançabilité sans être trop péssimiste
    Trappel TP sur l'analyse d'ordonnancement avec AADL:

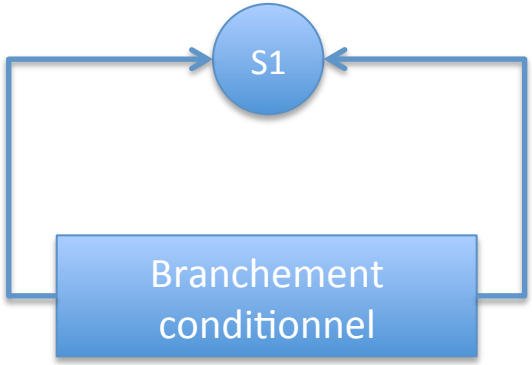
Tâche 1



computation(2 ms);
pos_x !<;
pos_y !<;
computation(5ms);
pos_x !>;
pos_y !>;
computation(10ms)

Tâche 2

computation(10 ms);
pos_y !<;
pos_x !<;
computation(7 ms);
pos_y !>;
pos_x !>;
computation(3 ms)



computation(20 ms);
pos_y !<;
computation(4 ms);
pos_x !<;
computation(1 ms);
pos_y !>;
pos_x !>;
computation(10 ms)

- Ordonnançable? …

Erreur temporelle: pourquoi?

- WCET mal calculé (caches/multi-cœurs/pagination…)
  - en pratique, les industriels sont peu enclins à utiliser des WCET calculés pour le dimensionnement de leurs applications… Il préfèreront la simulation
- Blocking time: complexité vs. Pessimisme des configurations à analyser
  - nous avons vu au TP précédent qu'il peut être difficile d'isoler une configuration qui prouve l'ordonnançabilité sans être trop péssimiste
- Mauvaises hypothèses dans l'analyse d'ordonnancement
  - Cout des changements de contextes, démarrage synchrone
- Mauvaise interprétation des résultats théoriques
  - Conditions nécessaire et/ou suffisantes
- Changement de plate-forme d'exécution sans mise à jour des paramètres temporels
- …

Robustesse au non-respect des hypothèses

- Il se peut que le système soit capable de fonctionner
  - En supportant K erreurs temporelle toute les N exécutions consécutives d'une tâche. Dans ce cas un dépassement d'échéance est ignoré tant qu'il n'y a pas eu plus de K erreurs parmi N exécutions consécutives.
    C'est souvent le cas dans des applications de contrôle, à condition de bien choisir K et N.
  - En exécutant un sous-ensemble des fonctionnalités: certaines tâches sont moins « vitales » pour le fonctionnement du système. Par exemple, la *journalisation des évènements* peut-être interrompue pour laisser à un drone plus de CPU pour *assurer sa mission*.
  - En exécutant certaines tâches périodiques à une fréquence moins élevée que la fréquence nominale

Non robustesse aux erreurs temporelles: conséquences?

- L'application passe dans un *état de dysfonctionnement* pouvant entrainer la défaillance du système de l'échec d'une mission, de la perte d'équipements (conséquences économiques) jusqu'à la perte de vies humaines.

- Il faut donc être en mesure de
  - ⌘ Détecter ces erreurs temporelles
  - ⌘ Eviter leur propagation (zones de confinement)
  - ⌘ Mettre en œuvre des mécanismes de correction

Erreur temporelle: détection

- Dépassement d'échéance ratée (deadline miss)
- Dépassement de WCET (WCET overrun)
- Rafale d'événement (Sporadic overrun)
- Sur-utilisation des ressources partagées (resources overrun)

- Le support de la détection de dépassement d'échéance dépend du standard/système d'exploitation:
  - ARINC653: propose la détection de certaines erreurs logicielles dont le dépassement d'échéance des tâches. C'est donc de la responsabilité du noyau (et donc de l'ordonnanceur) de détecter les dépassements d'échéance.
  - RT-POSIX: aucun support proposé par la spécification; c'est donc de la responsabilité du développeur d'ajouter ce mécanisme.

- Pour le reste (autres erreurs), pas vraiment de support dans les OS… c'est donc de la responsabilité du développeur d'ajouter le mécanismes requis

Erreur temporelle: traitement

- Le confinement consiste à éviter la propagation de l'erreur
  - Ordonnancement hiérarchique par groupe de tâche de même niveau de criticité
  - Exemple: partitionnement temporel ARINC653

- La tolérance aux fautes du système est sa capacité à continuer de fonctionner correctement en présence d'erreurs
  - C'est le cas des applications non-impactées grâce au confinement (à condition qu'il n'y ai pas d'autres dépendances, de données par exemple)

- Le recouvrement vise à remettre le système dans son état de fonctionnement nominal

A propos du recouvrement

- L'erreur se manifeste lors d'une surconsommation du CPU
- La robustesse est assurée
  - Grâce au confinement
  - Ou en dégradant certaines fonctionnalités pour libérer le CPU
- Pour revenir dans les conditions nominales (*recouvrement*), il faut redonner du CPU aux applications dont le fonctionnement a été dégradé
  - Il est difficile de savoir quand on peut redonner le CPU sans provoquer une nouvelle faute temporelle
  - D'un autre coté, il est préférable de rétablir le fonctionnement nominal au plus vite pour une meilleure qualité de service
- En pratique, beaucoup d'effort de dimensionnement sur bancs d'essais, simulateurs, etc…

Exemple au niveau OS: ARINC653 Health Monitoring

Health Monitoring (HM)

- Le standard ARINC653 fournit une section dédiée au « health monitoring », c'est à dire à la détection et au traitement des erreurs.
  - Remarque: le HM couvre un périmètre beaucoup plus large que les erreurs temporelles. Nous ne donnons ici que des principes généraux avec un focus sur les fautes temporelles.

- Il s'appuie sur
  - Une table de configuration (fichier de configuration XML)
  - Quelques services fournis par l'OS

- Le HM définit trois niveaux d'erreurs: process, partition, et module.
  - Cela correspond au trois niveaux où des erreurs logicielles peuvent se produire ou être détectée.

HM ARINC653: la détection des erreurs

- Les erreurs sont détectées au niveau
  - ⌘ hardware (memory violation, stack/heap overflow, zero divide…)
  - ⌘ noyau (problème de configuration, **dépassements d'échéances**…)
  - ⌘ Applicatif (valeur inattendu, trop ancienne, appel système qui échoue…)

- La liste complète des erreurs qui peuvent être détectées pas un OS est « implementation specific »

HM ARINC653: le traitement des erreurs

- Le traitement d'une erreur dépend du niveau où elle se produit

- La table de configuration donne une correspondance entre le niveau de l'erreur, son type, et l'action à mener en cas d'occurrence.
  - Comme les types d'erreurs et les actions possibles sont dépendants de l'implémentation, le contenu de la table est aussi dépendant de l'implémentation.
  - Le standard donne cependant des exemples:
    - Ignorer l'erreur
    - Arrêter/Redémarrer le processus fautif
    - Arrêter/Redémarrer la partition fautive
    - Arrêter/redémarrer le module

HM ARINC653: types et services définis par le standard (1/2)

- Pour les erreurs au niveau applicatif, le standard définit des services et des types de données.

- Types de données
  - ERROR_CODE_TYPE, un enum: **DEADLINE_MISSED**, APPLICATION_ERROR, NUMERIC_ERROR, ILLEGAL_REQUEST, STACK_OVERFLOW, MEMORY_VIOLATION, HARDWARE_FAULT, POWER_FAIL
  - ERROR_STATUS_TYPE une structure qui contient :
    - Un champ ERROR_CODE de type ERROR_CODE_TYPE pour identifier la nature de l'erreur
    - MESSAGE de type ERROR_MESSAGE_TYPE, une chaine de caractères
    - LENGTH la longueur de MESSAGE
    - FAILED_PROCESS_ID de type PROCESS_ID_TYPE pour identifier le processus fautif
    - FAILED_ADDRESS : SYSTEM_ADDRESS_TYPE adresse d'occurrence de la faute

HM ARINC653: types et services définis par le standard (2/2)

- Pour les erreurs au niveau applicatif, le standard définit des services et des types de données.
- Les principaux services
  - RAISE_APPLICATION_ERROR, pour signaler une erreur depuis le code d'un process ou d'une partition.
  - CREATE_ERROR_HANDLER, pour créer un process ARINC qui est réveillé sur occurrence d'une erreur au niveau application. Par exemple, pour un dépassement d'échéance, la détection se fait au niveau noyau (dans l'ordonnanceur) mais c'est bien une erreur au niveau application.
  - GET_ERROR_STATUS, pour récupérer les informations relatives à l'erreur (message, processus et partition fautives, …)
- Le redémarrage d'une partition ou d'un processus se fera en utilisant les services de gestion des partitions et processus (SET_PARTITION_MODE, START, STOP, etc.)

Par rapport au cours précédent

- Dans le cours précédent, on vous a parlé de forward recovery et backward recovery

- Table → forward recovery seulement (pas de sauvegarde d'état)
  - Dans le cas de ARINC653, cela concerne des erreurs au niveau partition/module

- CREATE_ERROR_HANDLER → forward ou backward…
  - Dans le cas de ARINC653, cela concerne des erreurs au niveau applications

Quid de la gestion des erreur temporelles avec RT-POSIX?

RT-POSIX: erreurs de dépassement d'échéance

- Il n'y a pas de support pour détecter les dépassements d'échéance en RT-POSIX.

- Il est donc à la charge du développeur de développer un *watchdog*: un mécanisme de temporisation qui vérifie que chaque tâche terminent avant la date de sa prochaine échéance.

- Peut servir pour détecter des
  - ⌘ Dépassement d'échéance
  - ⌘ Dépassement de WCET

Idée générale

- En s'inspirant du pattern définit dans ARIN653, on peut créer une tâche *watchdog* de très forte priorité, avec un timer armé avec la date de la prochaine échéance. Exemple d'implémentation (que vous ferez en TP):
  - Chaque tâche arme un timer au début de son exécution, et le désarme à la fin de son exécution
  - Si la tâche n'a pas désarmé le timer au moment de sa deadline, un signal est émis
  - La tâche de watchdog est en attent et se réveille sur occurrence de ce signal

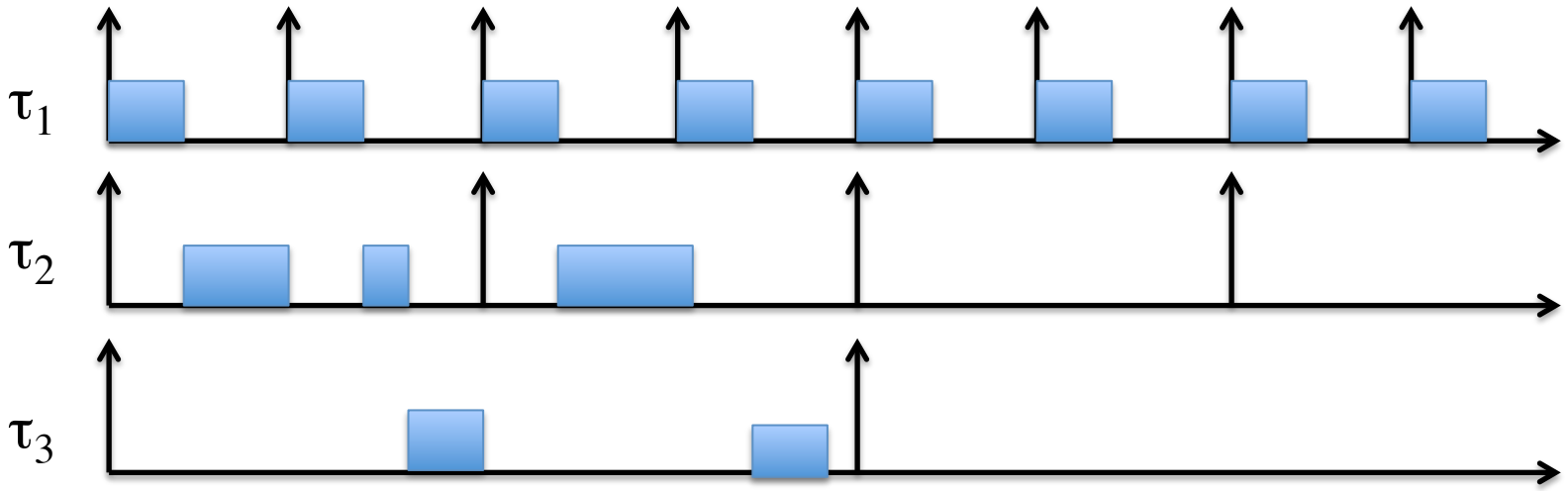
- Pb: comment analyser le surcoût?
  - Directement intégré au WCET des tâches

Exemple: schéma d'ordonnancement

- Exemple simple: ordo RMS, 3 tâches (τ_1, τ_2, et τ_3) et un verrou (v_1).

Tâche	Période	WCET	Deadline	Priorité
τ_1	5 ms	2 ms	5 ms	3
τ_2	10 ms	4 ms	9 ms	2
τ_3	20 ms	7 ms	17 ms	1

- Scénario d'ordonnancement pire cas sans la tâche *watchdog*

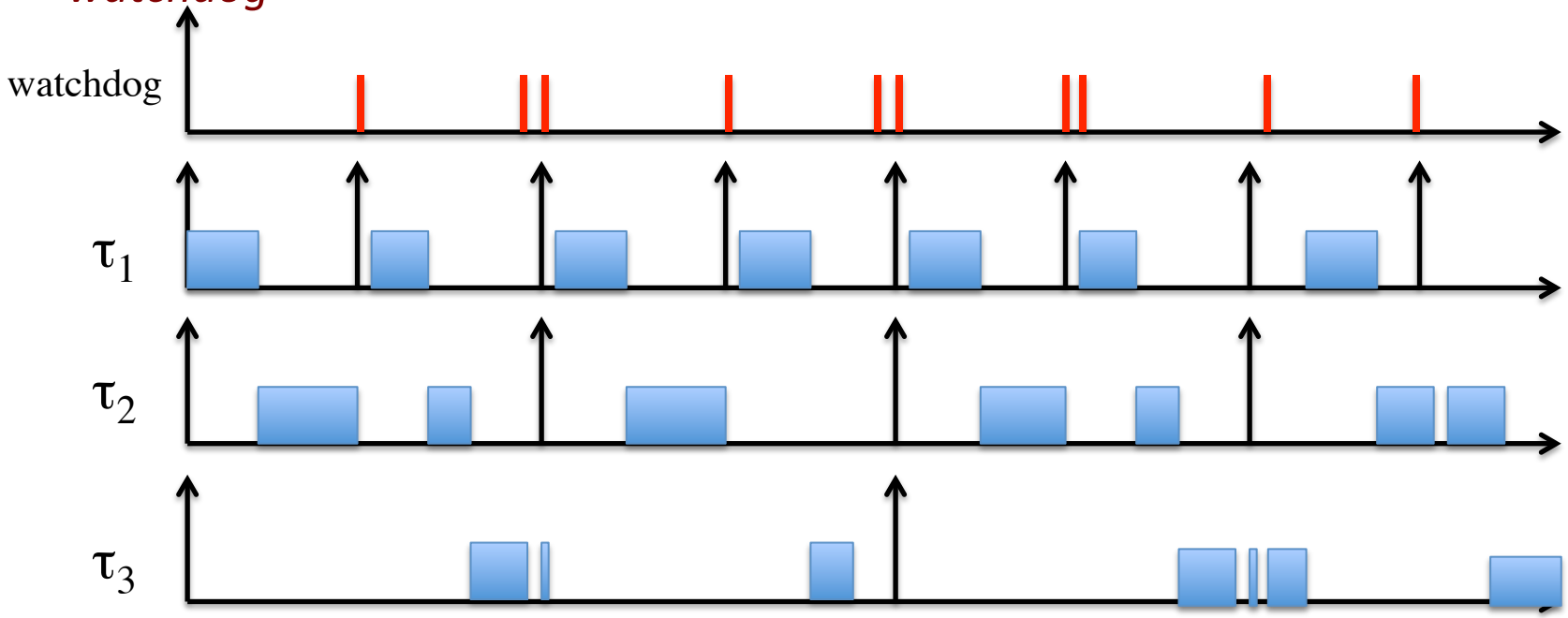


Exemple: schéma d'ordonnancement

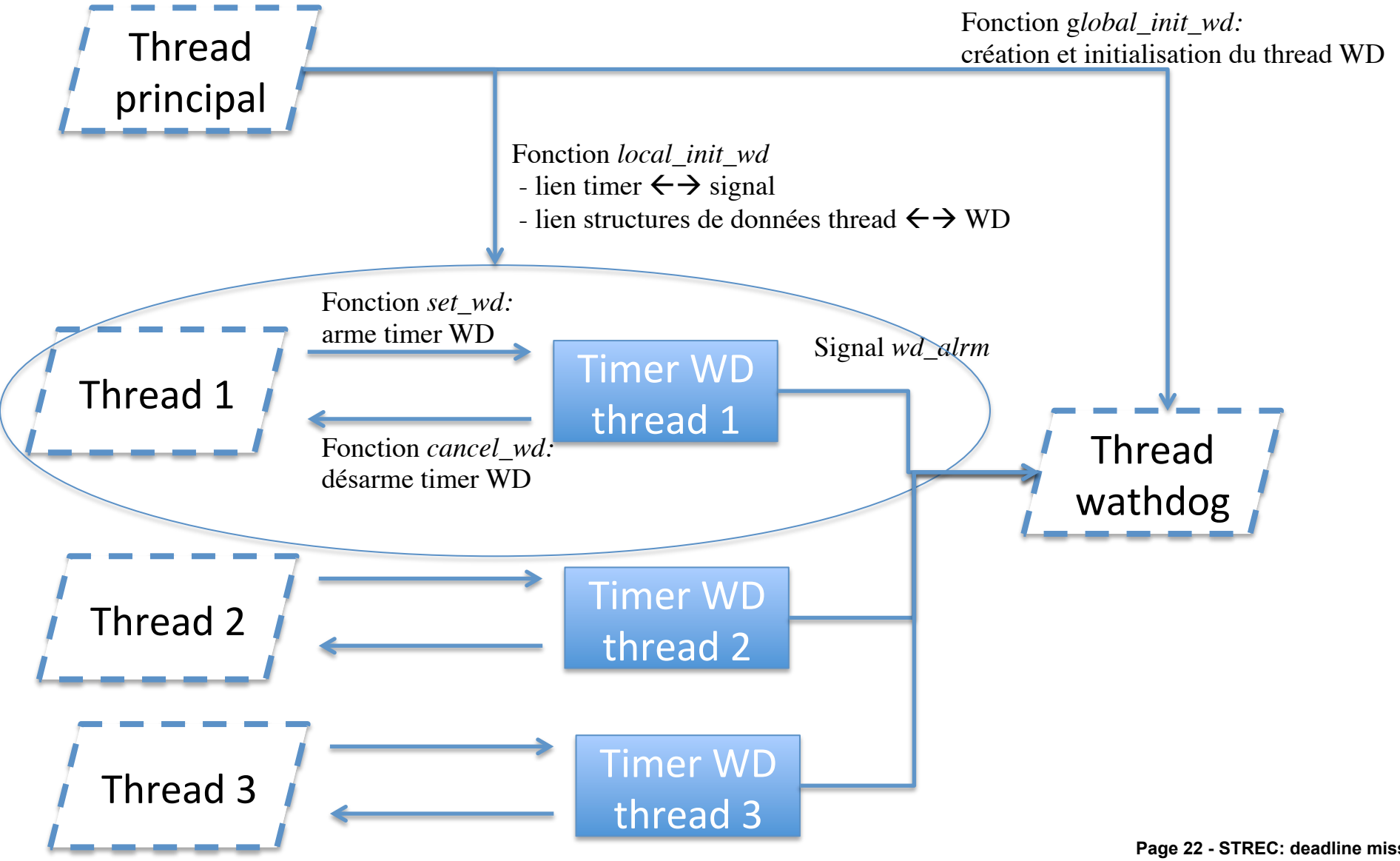
- Exemple simple: ordo RMS, 3 tâches (τ_1, τ_2, et τ_3) et un verrou (v_1).

Tâche	Période	WCET	Deadline	Priorité
τ_1	5 ms	2 ms	5 ms	3
τ_2	10 ms	4 ms	9 ms	2
τ_3	20 ms	7 ms	17 ms	1

- Scénario d'ordonnancement avec illustration des réveils potentiels de la tâche *watchdog*



Au niveau architecture (utile pour le TP)



Rappel, triptyque: détection/confinement/ recouvrement

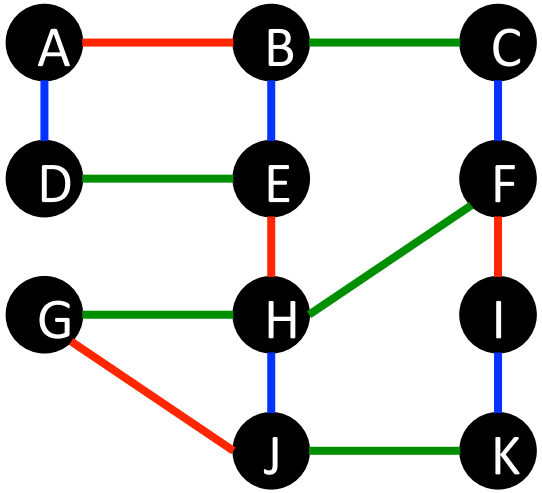
- Nous avons parlé de la détection en RT-POSIX et du triptyque pour ARINC653
- Difficulté en RT-POSIX : confinement ou recouvrement → nécessite de modifier la politique d'attribution du CPU
- Souvent, cela se fait via un mécanisme plus général: les modes de fonctionnement
  - ⌘ Cette notion peut également se retrouver comme un mécanisme de recouvrement en ARINC653

Modes de fonctionnement
(réponse classique à l'apparition d'erreurs)

Modes de fonctionnement

- Définition: un mode de fonctionnement représente un état du système dans lequel celui-ci fournit un sous-ensemble de ses fonctionnalité.

- Exemple: Aller de C à J dans la carte (initialement le robot est sur un point noir)
  - ⌘ Calculer un chemin (séquence de couleurs)
  - ⌘ Chercher la prochaine ligne à suivre
  - ⌘ Détecter l'atteinte de la ligne à suivre
  - ⌘ Suivre une ligne de couleur
  - ⌘ Détecter l'atteinte d'un point noir
  - ⌘ Détecter des obstacles sur la trajectoire
  - ⌘ Arrêter le robot en cas d'obstacle



Fonctionnalités par mode

- Modes
  1. Calculer un chemin
  2. Chercher la prochaine ligne
  3. Suivre une ligne
- Fonctionnalités par mode
  1. Calculer un chemin (séquence de couleurs).
  2. Chercher la prochaine ligne à suivre, Détecter l'atteinte de la ligne, Détecter des obstacles sur la trajectoire, Arrêter le robot en cas d'obstacle.
  3. Suivre une ligne de couleur, Détecter l'atteinte d'un point noir, Détecter des obstacles sur la trajectoire, Arrêter le robot en cas d'obstacle.

Architecture par mode

- Mode compute path: 1 tâche de calcul de la séquence de lignes à suivre.
- Mode follow line: 4 tâches
  - suivi de ligne (avec computePID)
  - détection d'atteinte d'un point noir (changement de mode)
  - détection d'osbtacle
  - MàJ de la couleur (tâche background requise par la plate-forme d'exécution)
- Mode search next line: 4 tâches
  - suivi de point noir
  - détection d'atteinte de la prochaine ligne (changement de mode)
  - détection d'osbtacle
  - MàJ de la couleur (tâche background requise par la plate-forme d'exécution)

Transitions de mode

- Compute path → Search next line
  - ⌘ Une fois le chemin calculé, on cherche la première ligne à suivre
- Search next line → Mode follow line
  - ⌘ Une fois la ligne atteinte, on la suit
- Mode follow line→ Search next line
  - ⌘ Une fois le point noir atteint, on cherche la prochaine ligne

Les modes en AADL

- Les modes permettent de modéliser la reconfiguration du système en fonction d'événements
  - définis au niveau des composants
  - seuls les événements (event ports) peuvent déclencher un changement de mode
  - certains sous-composants, connexions, etc. ne sont activés que dans certains modes
  - les valeurs des propriétés peut dépendre du mode courant
- Les changements de modes permettent de représenter des architectures (pseudo) dynamiques
  - évolution (dynamique) de la configuration de l'architecture
  - ensemble déterminé (statiquement) de configurations possibles

Exemple de modes

```
thread execution_thread
end execution_thread;

process a_process
features
  multi_thread : in event port;
  mono_thread : in event port;
end a_process;

process implementation a_process.configurable
subcomponents
  thread_1 : thread execution_thread;
  thread_2 : thread execution_thread in modes (multitask);
modes
  monotask : initial mode;
  multitask : mode;
  monotask -[ multi_thread ]-> multitask;
  multitask -[ mono_thread ]-> monotask;
end a_process.configurable;
```

Code AADL (exemple du robot)

```
process implementation proc.impl
    subcomponents
        backgroung_inst : thread backgroung.impl in modes (follow_line_mode,
                                                           search_next_line_mode);
        follow_line_inst : thread follow_line.impl in modes  (follow_line_mode);
        color_lookup_inst : thread color_lookup.impl in modes   (follow_line_mode);
    …
    modes
        follow_line_mode : mode;
        compute_color_mode : initial mode;
        search_next_line_mode : mode;
        follow_line_mode -[color_lookup_inst.reached]-> compute_color_mode;
        compute_color_mode -[compute_next_color_inst.computed]->
                              search_next_line_mode;
        search_next_line_mode -[search_next_line_inst.found]-> follow_line_mode;
    …
end proc.impl;
```

Mode pour tolérer des dépassements d'échéance

- Le but est de libérer temporairement le CPU
  - ⌘ Réduire la fréquence des tâches → attention à l'intégrité des buffers partagées, par ex. calcul de vitesse à partir de positions…
  - ⌘ Désactiver des tâches → attention à ne pas les désactiver brutalement alors qu'elles sont dans une section critique…

Sémantique du changement de mode

- D'un point de vu « temps-réel », les changements de modes posent quelques difficultés:
  - S'assurer que les tâches dans le mode cible sont ordonnançable malgré la surconsommation CPU liée au changement de mode.
  - S'assurer qu'aucune tâche impactée par le changement de mode n'est dans une section critique au moment du changement de mode.
  - Préserver la cohérence de flux de données.
  - S'assurer que le mode cible est atteint suffisamment rapidement
- Certains de ces objectifs sont en conflits… Différents protocoles de changement de mode devront donc être proposés.

Protocoles proposés par AADL

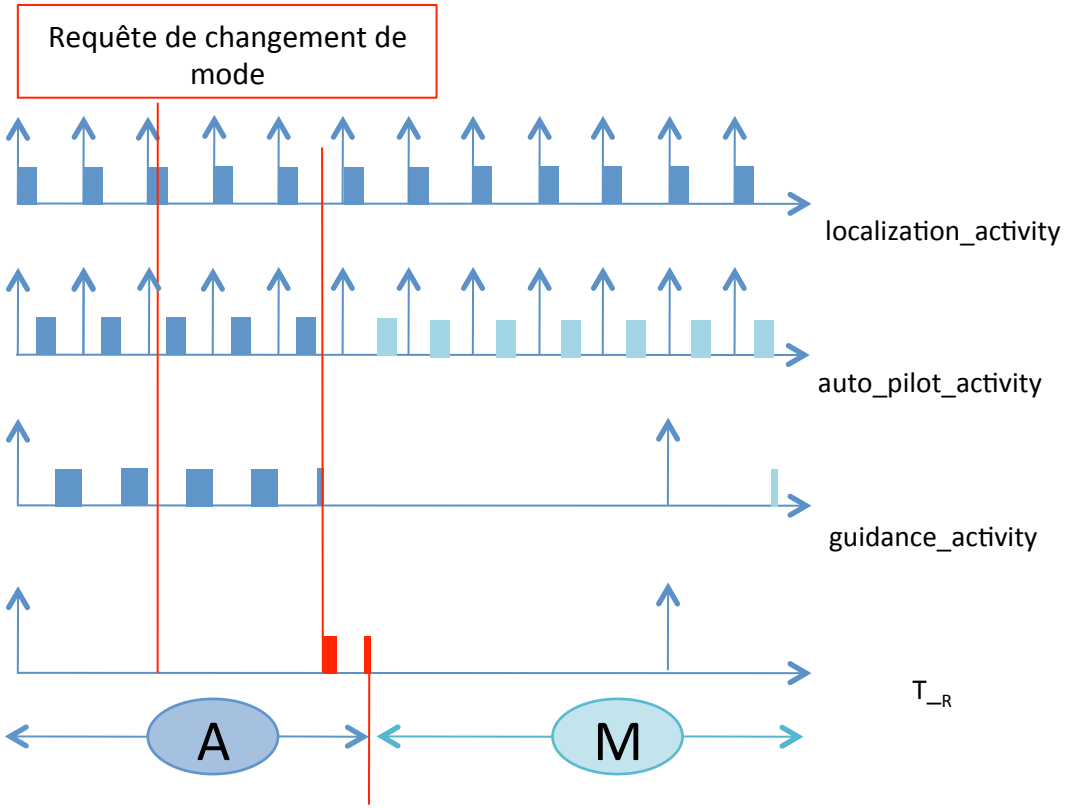
- A mode transition may actually be performed immediately if it is considered an *emergency*, or it may be performed in a *planned* fashion after currently executing threads complete their execution and the dispatches of a set of synchronized thread are aligned. This is indicated by the Mode_Transition_Response property on the mode transition with the default being planned.
- Deactivated threads transition to the *suspended awaiting mode* state.
- A component is considered synchronized if it has a *Synchronized_Component* property value of true.

Patron d'implémentation

- Mode = variable globale
  - ⌘ mise à jour par une « tâche de reconfiguration »
  - ⌘ Donnée consultée par les « tâches impactées »
- Tâche impactée : tâche dont le flux d'exécution varie en fonction du mode
  - ⌘ Nécessite un verrou de type lecteur/écrivain (tâches impactées = lecteur; tâche de reconfigucation = écrivain)
- Algorithme de protection (tâche impactée)
  - ⌘ Acquisition du verrou en lecture
  - ⌘ Exécution du code de la tâche
  - ⌘ Libération du verrou

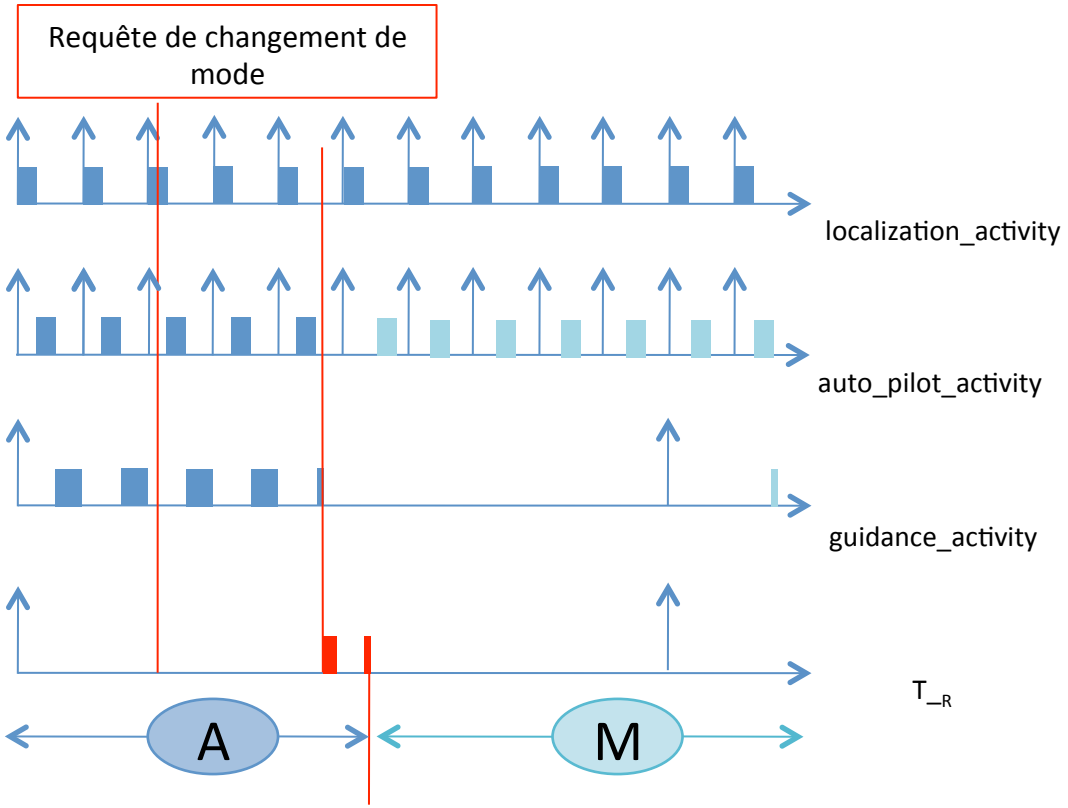
Protocole P1: spécification AADL et chronogramme

- Mode_Transition_Response => planned
- Ensemble des tâches synchronisées = ∅



Protocole P1: implémentation

- la priorité de la tâche de reconfiguration est inférieur à la priorité des tâches impactées
- Le reste de l'implémentation a été spécifié plus haut (voir page « patron d'implémentation »)
  - ⌘ Lecteur/écrivain
  - ⌘ Code = Lock/exécute/unlock

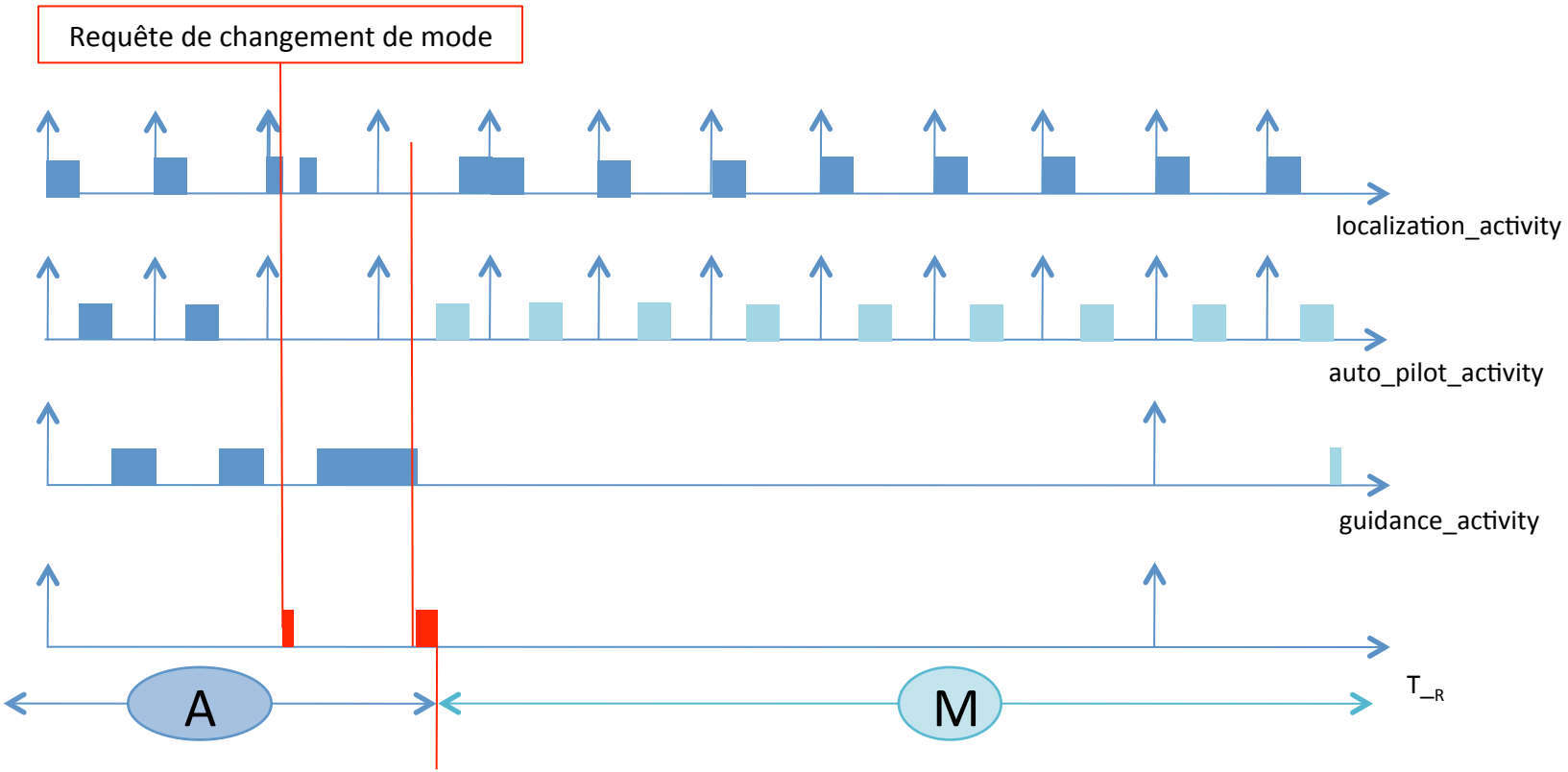


Protocole P1: analyse

- Objectif : favoriser l'application en minimisant les perturbations que subissent les tâches
- Temps de changement de mode?
- Ordonnançabilité en mode source, destination, et transition?

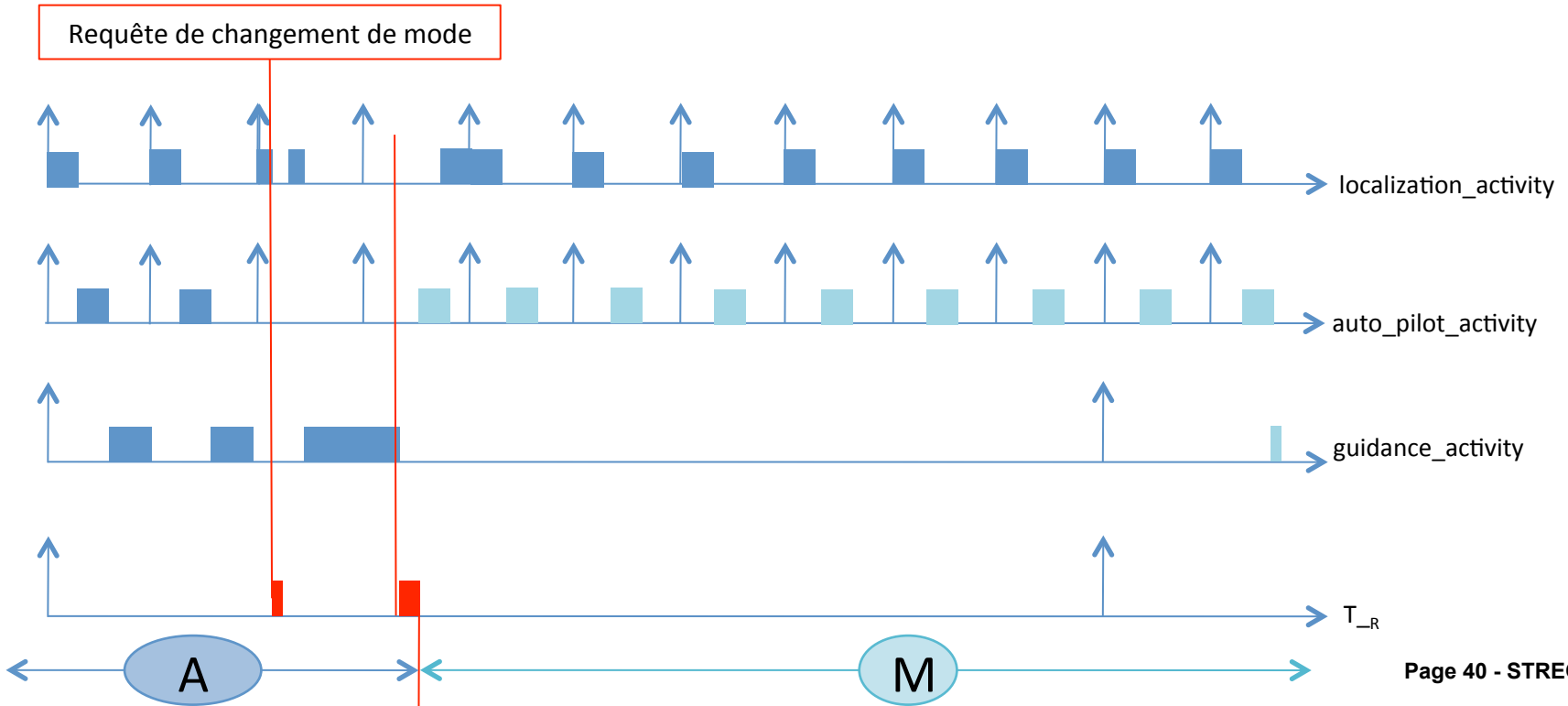
Protocole P2: spécification AADL et chronogramme

- Mode_Transition_Response => emergency
- Ensemble des thread critiques = ∅



Protocole P2: implémentation

- Principe
  - Tâche de reconfiguration plus prioritaire que les tâches impactées
  - RWLock « à la PCP »: la priorité des tâches bloquant la reconfiguration hérite d'une priorité plafond
- Le reste de l'implémentation a été spécifié plus haut (voir page « patron d'implémentation »)
  - Lecteur/écrivain
  - Code = Lock/exécute/unlock

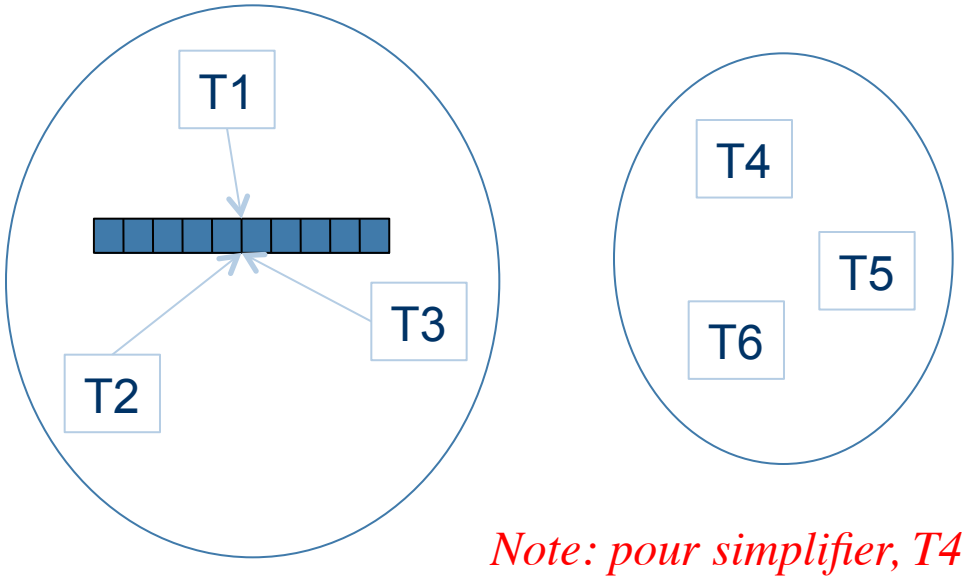


Protocole P2: analyse

- Objectif: accélérer la reconfiguration en précipitant la fin des tâches impactées
- Temps de changement de mode?
- Ordonnançabilité en mode source, destination, et transition?

Protocole P3: spécification AADL

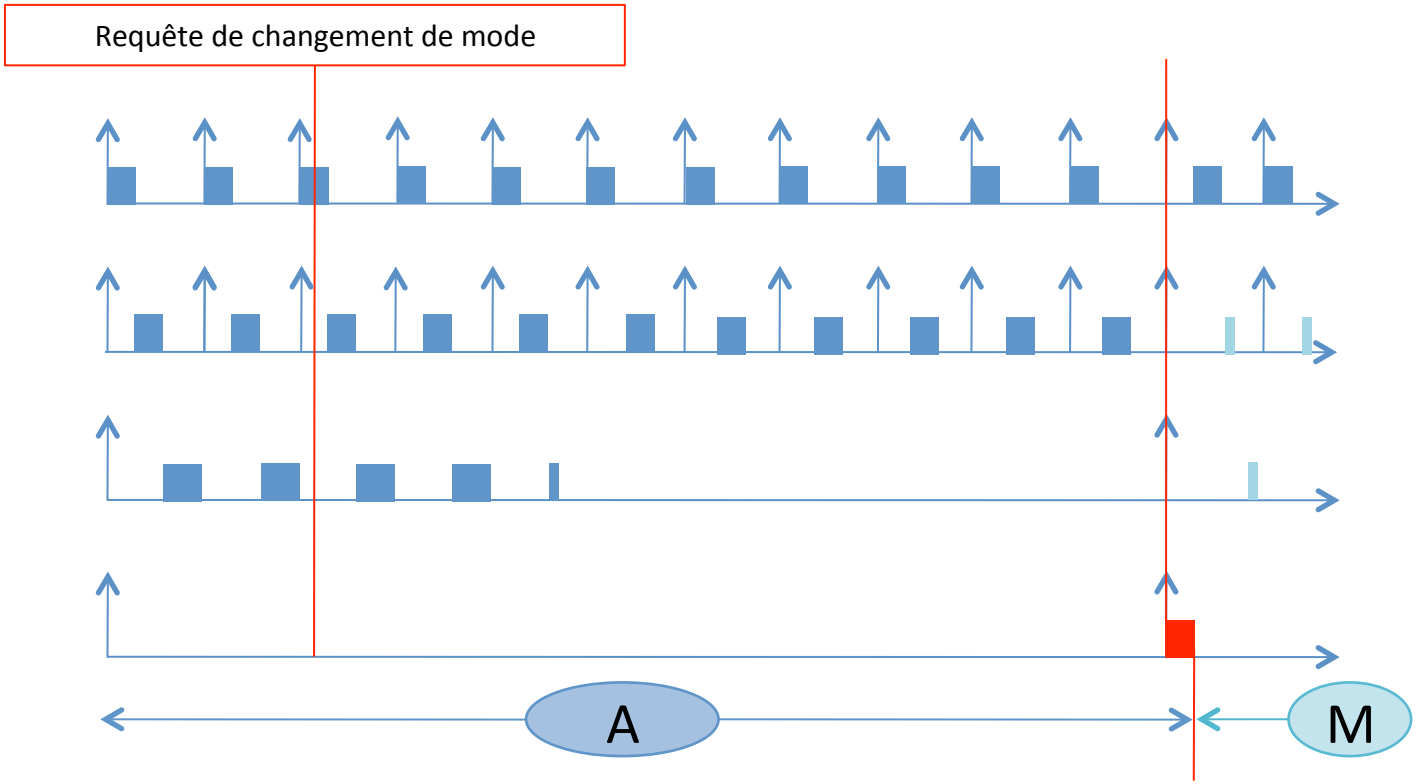
- Mode_Transition_Response => planned
- Ensemble des tâches synchronisées = {T1, T2, T3}
  - Car T1 T2 et T3 partagent un accès à un buffer dont ils produisent ou consomment les valeurs



Note: pour simplifier, T4 T5 et T6 ne sont pas présentes dans les chronogrammes qui suivent

Protocole P3: spécification AADL et chronogramme

- Mode_Transition_Response => planned
- Ensemble des tâches synchronisées = T1, T2 et T3
- La synchronisation avec T4, T5 et T5 se fait en suivant le protocole P1
- Conséquence: T1, T2 et T3 ne repartiront que quand T4 T5 et T6 auront fini



Protocole P3: implémentation

- Une tâche de reconfiguration de forte priorité (plus forte que les tâches impactées), périodique, de période = hyperpériode des tâches impactées

- Le reste de l'implémentation a été spécifié plus haut (voir page « patron d'implémentation »)
  - Lecteur/écrivain
  - Code = Lock/exécute/unlock

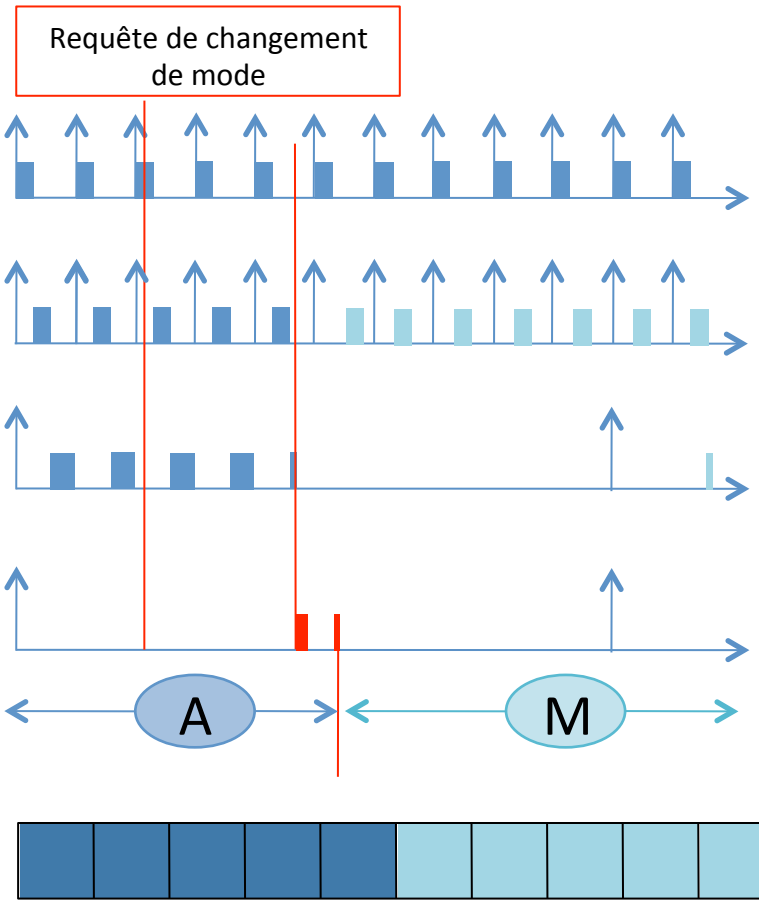
Protocole P3: analyse

- Objectif 1: préserver l'intégrité d'un ensemble de données en retardant la reconfiguration jusqu'à ce que l'hyper-période soit atteinte
- Objectif2: favoriser les tâches non-synchronisées en minimisant les perturbations que subissent ces tâches

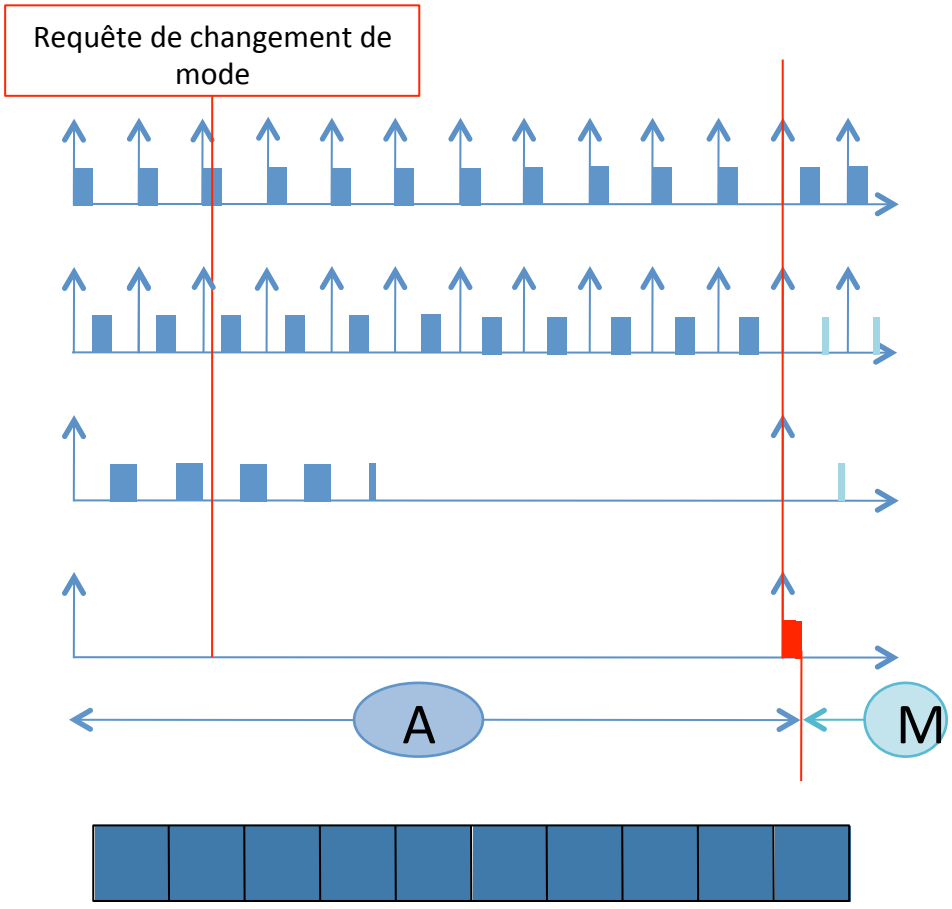
- Temps de changement de mode?

- Ordonnançabilité en mode source, destination, et transition?

Comparaison P1 et P3: flux de données



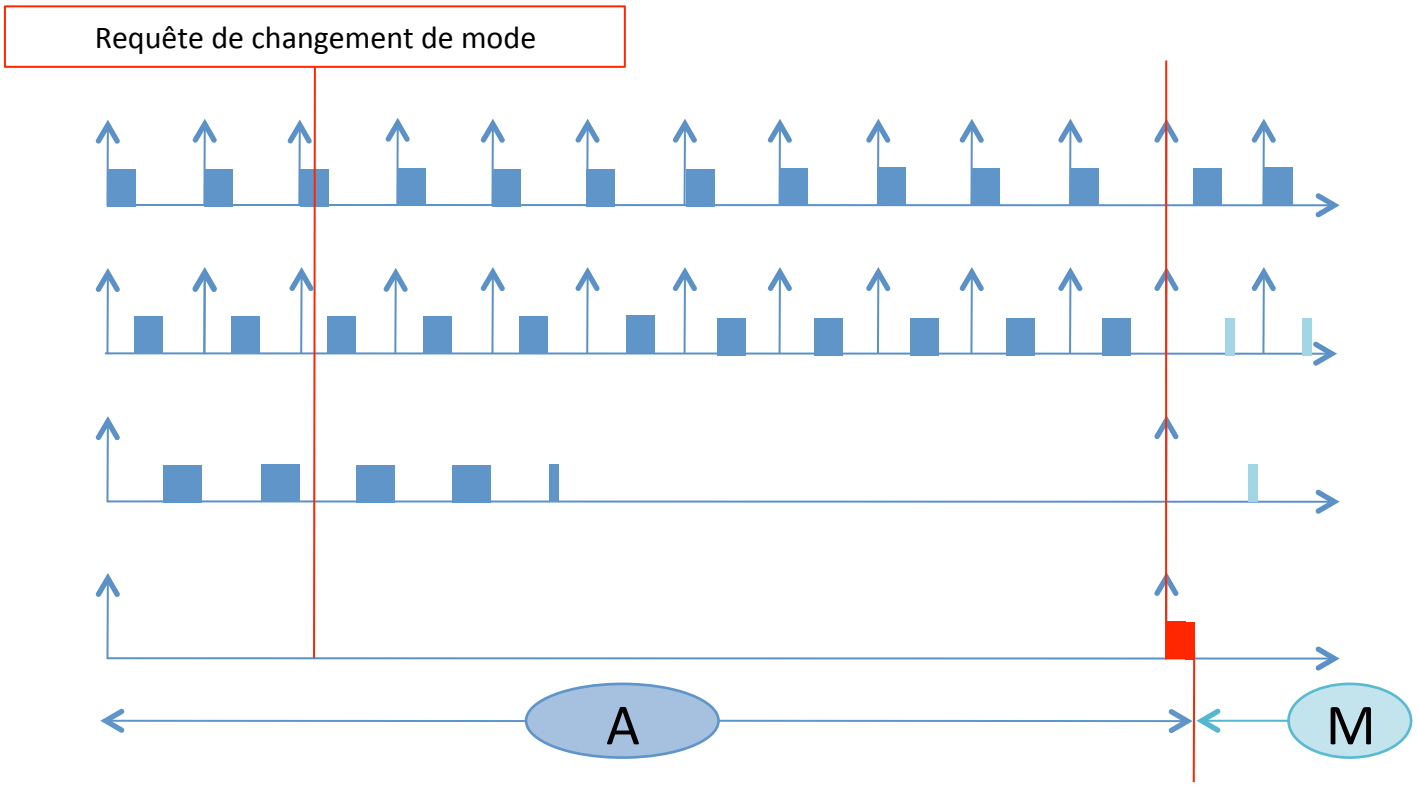
Ensemble de données incohérentes



Ensemble de données cohérentes

Protocole P4: variante de P3hors périmètre AADL …

- Ensemble des tâches synchronisées = T1, T2 et T3
- La synchronisation avec T4, T5 et T5 se fait en suivant le protocole P1
- Conséquence: T1, T2 et T3 ne repartiront que quand T4 T5 et T6 auront fini mais celles ci sont configurées avec EMERGENCY (version PCP)



Protocole P4: implémentation

- 1 tâche de reconfiguration périodique de plus forte priorité que les tâches synchronisés et impactées → désactive les tâches synchronisées et précipite la fin des tâches non-synchronisées (avec un lecteur écrivain avec PCP)

- Le reste de l'implémentation a été spécifié plus haut (voir page « patron d'implémentation »)
  - Lecteur/écrivain
  - Code = Lock/exécute/unlock

Protocole P4: analyse

- Objectif 1: préserver l'intégrité d'un ensemble de données en retardant la reconfiguration jusqu'à ce que l'hyper-période soit atteinte
- Objectif2: favoriser les le changement de mode plutôt que les tâches non-synchronisées

- Temps de changement de mode?

- Ordonnançabilité en mode source, destination, et transition?

Comparaison des protocoles

Critère \ Protocole de reconfiguration	P1	P2	P3	P4	Validation
Ordonançabilité	++	--	++	-	analyse
Rapidité	-	++	--	-	expérimentations
Cohérence des données	+	+	++	++	Analyse/ expérimentations

P1 : protocole privilégiant l'application
P2 : protocole privilégiant la reconfiguration
P3 : protocole garantissant la cohérence des données (sans PCP)
P4 : protocole garantissant la cohérence des données (avec PCP)

Conclusion

- La détection d'erreurs (temporelles) peut avoir un impact non négligeable sur l'architecture
  - 1 thread ajouté
  - Timers logiciels
- Il en est de même des modes de fonctionnement
  - Potentiellement plusieurs threads (en fonction des changements de modes et de leurs protocoles)
- Les traitements/contraintes associés à ces mécanismes sont fortement dépendants de l'application, de choix de conception, du type de faute…
  - Mécanismes OS complexes, souvent en interaction
  - Beaucoup de mise au point/simulation…
- Ils sont incontournables dans les systèmes temps-réel (critiques) car même si le système est faiblement critique c'est la qualité de service qui est en jeu